$u\gamma - Golomb$: A new Golomb Code Variation for the Local Bernoulli Model

Ilias Nitsos[1], Georgios Evangelidis[1], and Dimitris Dervos[2]

[1] Department of Applied Informatics, University of Macedonia
156 Egnatia Str., 54006 Thessaloniki, Greece
{nitsos, gevan}@uom.gr
[2] Department of Information Technology, TEI
P.O. Box 14561, 54101 Thessaloniki, Greece
dad@it.teithe.gr

Abstract. Several compression codes exist today that have been developed to reduce the size of inverted file indexes used widely in information retrieval implementations targeting text databases. In the present study, we examine some of the most significant integer compression codes and propose $u\gamma - Golomb$, a variation of the most popular scheme: the Golomb code for the local Bernoulli model. As a test-bed, we utilize text databases from the TREC collection that take up almost 1GB on disk. The proposed new variation does not introduce any additional computational overhead when it is compared to the original compression code. With regard to storage utilization efficiency, experimental results reveal a considerable improvement over the original compression code in the case of non-uniform text databases.

1 Introduction

Several schemes have been developed for indexing text databases, during the past decades. The most popular, nowadays, are the signature files [4], [5] and the inverted files [6], [17], [16]. In the case of inverted files, much progress has been made, due to great advances in the field of integer compression codes [9], [3], [8], [11], [1], [12]. Several such codes exist that allow the average pointer (integer) inside an inverted file to be stored in less than 1 byte [16], thus producing very compact indexes. Compressed indexes are also fast, because fewer disk accesses are required and the decompression CPU cost is low [15].

In the present paper we consider a number of compression codes and focus mainly on the most popular one, the Golomb code for the local Bernoulli model [11], [8], [17], [12]. We introduce $u\gamma - Golomb$, a new variation for the latter, one that produces similar results in the case of uniform text databases and improves the compression in the case of non-uniform text databases. A text database consisting of many documents is characterized as uniform or non-uniform depending on whether the words are scattered uniformly in its documents or not.

In Section 2 we examine several codes that have been developed for compressing integers inside the inverted files and the models they are based on. A new Golomb code variation, $u\gamma - Golomb$ is proposed in Section 3. The performance of the new variation in the case of some selected TREC collections is considered in Section 4. Section 5 concludes on the proposed variation and includes issues for further research.

2 Codes for Compressing Inverted Files

A very efficient index structure used in full-text retrieval systems is the inverted file index [6], [17], [16]. Let us suppose that we have a text database consisting of N documents, each represented by a positive integer $d \in [1, \ldots, N]$. For each distinct word t, an inverted list is created that stores all the document numbers d containing t. All those inverted lists are stored in a single file, known as the inverted file [6].

The compression codes presented in this section are some of the most popular for compressing the integers that are stored in an inverted file and utilize run length encoding [8]. Instead of storing the absolute numbers d of the documents containing a word, inverted lists store their differences. For example, instead of storing the d list $\{2, 9, 10, 15, 16, 20\}$, for word t, we could store the corresponding d-gap list $\{2, 7, 1, 5, 1, 4\}$. In general, this results in storing smaller and more frequently occurring integers. The initial list is easily reconstructed from the d-gap list.

The codes discussed in the subsections that follow are based on models that take into consideration the probability distribution of d-gap sizes and assign small bit codes to frequent d-gap sizes and larger bit codes to rare ones. Depending on whether each inverted list in the inverted file is compressed using the same (global) or different (local) parameters, the models and their codes are categorized as non-parameterized and parameterized, respectively. The interested reader should consult [16] for a detailed discussion.

2.1 Non-parameterized codes

The most popular non-parameterized codes are:

Unary code: In the unary coding scheme each positive integer x is represented by x-1 "one" bits followed by a "zero". For example, the number 5 in unary will be stored as 11110. A list of the unary codes for the first ten positive integers is shown in Table 1.

The unary code is equivalent to assigning a probability of $\frac{1}{2^x}$ to gaps of length x [16].

Elias γ code: This is a code that achieves significant savings by encoding each positive integer as follows [3]:

- store number $1 + \lfloor log_2 x \rfloor$ in unary,
- store the remainder $x - 2^{\lfloor log_2 x \rfloor}$ in binary using $\lfloor log_2 x \rfloor$ bits.

Table 1. Examples of codes for integers

		Golomb				
x unary	γ code	b=2	b=3	b=4	b=6	b=7
1 0	0	00	00	000	000	0000
2 10	100	01	010	001	001	0001
3 110	101	100	011	010	0100	0010
4 1110	11000	101	100	011	0101	0011
5 11110	11001	1100	1010	1000	0110	0100
6 111110	11010	1101	1011	1001	0111	0101
7 1111110	11011	11100	1100	1010	1000	0110
8 11111110	1110000	11101	11010	1011	1001	10000
9 111111110	1110001	111100	11011	11000	10100	10001
10 1111111110	1110010	111101	11100	11001	10101	10010

A list of the γ codes for the first ten positive integers is shown in Table 1. The γ code is equivalent to assigning a probability of $\frac{1}{2x^2}$ to gap x [16].

Golomb code for the global Bernoulli model: Let us suppose that we have an N document text database that contains n distinct words and f index pointers (i.e., f distinct *"document, word"* pairs).

The global Bernoulli model assumes that large text databases are uniform, i.e. the words are distributed uniformly across the N documents. This, in turn, implies that the probability of any one randomly selected word to appear in any one randomly selected document is $p = f/(N \times n)$. It has been shown that the d-gaps in this case can be efficiently represented by the Golomb code [8]. According to this code, for a given parameter b, each positive integer x is encoded as follows:

- store number $q + 1$ in unary, where $q = \lfloor (x - 1)/b \rfloor$,
- store the remainder $r = x - q \times b - 1$ in binary using either $\lfloor log_2 b \rfloor$ or $\lceil log_2 b \rceil$ bits.

A list of Golomb codes for the first ten positive integers, for some values of the parameter b, is shown in Table 1. At the decompression phase, the original number is computed as $x = r + q \times b + 1$. For a given value of p, b is calculated as follows: $b = \lceil \frac{log_2(2-p)}{-log_2(1-p)} \rceil$ [7].

2.2 Parameterized codes

The most popular parameterized codes are:

Golomb code for the local Bernoulli model: The difference between the global and the local Bernoulli model is that in the latter each list is associated with a different b parameter [2]. The value of the parameter for the d-gap list of word t is calculated as $b = \lceil \frac{log_2(2-p)}{-log_2(1-p)} \rceil$, with $p = f_t/N$, where f_t is the number

of documents in the text database containing t. This means that for each distinct word t in the text database, the value for f_t must also be stored alongside with the inverted list of the word for the b parameter to be computed at decoding time. The value for f_t is stored at the head of each list, using γ code [16]. During the inverted list decompression, the f_t value is decoded first, then the b parameter is calculated and the decompression continues with the rest of the list.

Golomb code for the skewed Bernoulli model: The local Bernoulli model assumes that every word is scattered uniformly across the N documents. When indexing actual text collections, a lot of words tend to appear many times in some parts of the collection and not so often in other parts. A compression code that could be applied in this situation and produce more compact indexes, is the Golomb code for the skewed Bernoulli model [14], [11]. Nevertheless, in the case of inverted lists involving uniformly distributed d-gaps, the latter is slightly outperformed by the plain local Bernoulli model [16].

Other coding methods: Many other models and methods have been developed for coding gaps, like the local hyperbolic model that has low decoding performance [13] and the interpolative method [10] that takes advantage of the locality inside the inverted lists. Moreover, new algorithms have been developed recently that reorder the documents in a collection, in order to create high localities inside the inverted lists. The interpolative method utilizes the latter to achieve maximum compression [1].

The Golomb code for the local Bernoulli model is the preferred scheme because it combines acceptable compression, fast encoding/decoding, and simplicity of implementation [16], [17], [12]. For each inverted list, it is only the value for f_t that is stored in the head of the list and this value is also used for implementing ranking. The skewed Bernoulli model and the interpolative method produce more compact indexes in the case of non-uniform databases but they are slightly outperformed by the Golomb code for the local Bernoulli model in the case of uniform text databases [16]. Moreover, these models do not involve storing the values for f_t in each inverted list. Thus, additional space would be required in this case, in order to implement ranking.

3 $u\gamma - Golomb$

In this section we introduce $u\gamma - Golomb$, a new variation of the Golomb code for the local Bernoulli model. The aim is to improve the performance of this popular encoding method in the case of non-uniform text databases. The gain comes without producing larger indexes or introducing any additional computational overhead in the case of uniform text database collections. The main idea behind this variation is to reduce the number of bits used to store large values of q.

The plain Golomb code, for a given q, stores $q+1$ in unary and the remainder r in binary. When q becomes excessively large, a common situation for non-uniform collections, the unary code is not suitable because it uses many bits to store $q + 1$. The use of γ code should be preferred for large values of q, instead of the unary code.

We select a threshold value of q, let us call it q_0. During the coding process, when q is less than or equal to q_0 we store $q+1$ in unary (i.e., we use plain Golomb coding). On the other hand, when q is greater than q_0, instead of encoding $q+1$ in unary, we encode q using γ code. In this case, a constant number of "ones" is prefixed to the γ code representation, so that there is no confusion during the decoding process as to whether the value being read is $q+1$ encoded in unary code or q encoded in γ code. The following lemma determines the necessary "ones" that have to be prefixed.

Lemma 1. *If q_0 is the selected threshold value, then at least $q_0+1-\lfloor log_2(q_0+1)\rfloor$ "ones" must be prefixed to the γ code representation of q, for $q > q_0$.*

Proof. Let u_1 be the number of leading "ones" in the largest unary representation of q. This occurs when $q = q_0$, in which case we store q_0+1 in unary, thus $u_1 = q_0$. Let γ_1 be the number of leading "ones" in the smallest γ-code representation of q. This occurs when $q = q_0 + 1$, thus $\gamma_1 = \lfloor log_2(q_0 + 1)\rfloor$.

Clearly, in order to distinguish between the above two representations we must prefix x "ones" to the γ-code representation so that $\gamma_1 + x > u_1$. Thus, $x > u_1 - \gamma_1$ or $x > q_0 - \lfloor log_2(q_0 + 1)\rfloor$. The smallest x that satisfies the above condition is $x = q_0 - \lfloor log_2(q_0 + 1)\rfloor + 1$. □

The codes generated by this method for integers $[1 \ldots 20]$ when $q_0 = 4$ and $b = 2$ are shown in Table 2.

Table 2. $u\gamma - Golomb$ (b=2), for $q_0 = 4$

x	**code**	x	**code**	x	**code**	x	**code**
1 ($q = 0$)	00	6 ($q = 2$)	1101	11 ($q = 5$)	111,11001,0	16 ($q = 7$)	111,11011,1
2 ($q = 0$)	01	7 ($q = 3$)	11100	12 ($q = 5$)	111,11001,1	17 ($q = 8$)	111,1110000,0
3 ($q = 1$)	100	8 ($q = 3$)	11101	13 ($q = 6$)	111,11010,0	18 ($q = 8$)	111,1110000,1
4 ($q = 1$)	101	9 ($q = 4$)	111100	14 ($q = 6$)	111,11010,1	19 ($q = 9$)	111,1110001,0
5 ($q = 2$)	1100	10 ($q = 4$)	111101	15 ($q = 7$)	111,11011,0	20 ($q = 9$)	111,1110001,1

Let us examine these codes in detail. When $x \in [1 \ldots 10]$, the codes are the plain Golomb codes for $b = 2$ (Table 1), because for these values $q \leq q_0$. When $x \in [11 \ldots 20]$, then $q > q_0$, and the actual value of q is stored using γ code (shown between the two commas). The value after the second comma is the value for r stored in binary. The first $q_0 + 1 - \lfloor log_2(q_0 + 1)\rfloor = 3$ "ones" that appear before the first comma are necessary, so that when at least $q_0 + 1 = 5$ consecutive "ones" are encountered in the beginning of a code at decoding time, the first $q_0+1-\lfloor log_2(q_0+1)\rfloor = 3$ "ones" are ignored and what follows is treated as the γ code for q.

Several variations can be created by assigning different values to q_0. When q_0 is small, more values of q are being stored using γ code plus a constant prefix of $q_0 + 1 - \lfloor log_2(q_0 + 1)\rfloor$ "ones", and as q_0 increases the number of q values stored in γ code decreases.

4 Experimental Results

The document collection used as a testbed was the fifth volume of the TREC collection. The volume contains 475MB of articles from the Los Angeles Times (LAT), published during the January 1, 1989 – December 31, 1990 period, plus 470MB of documents of the Foreign Broadcast Information Service (FBIS) from 1994. The profiles for the LAT and FBIS collections as well as of their concatenation (LATFBIS) are shown in Table 3.

Table 3. Testbed collections' profiles

Collection	Size	Distinct words (n)	Total documents (N)	Total pointers (f)
LAT	475MB	288823	130472	31193292
FBIS	470MB	247563	131167	34982316
LATFBIS	945MB	437864	261639	66175608

We compared the following four codes:

- Elias γ code for all the gaps in all the lists of the inverted file. This code will be referred to as "$\gamma - Code$".
- Golomb code for the local Bernoulli model. An inverted list is created for every word in the collection. Each list also stores the value for f_t (document frequency for term t) using γ code. The f_t overhead for each word is included in the final results. This code will be referred to as "*Golomb*".
- Golomb code for the local Bernoulli model with $q + 1$ encoded using γ code instead of unary code. The f_t overhead for each word is included in the final results. This code will be referred to as "$\gamma - Golomb$".
- The Golomb code for the local Bernoulli model variation we propose (see Section 3). The f_t overhead for each word is once again included in the final results. This code will be referred to as "$u\gamma - Golomb$".

Figures 1 and 2 illustrate the results obtained for the LAT, FBIS, and LATFBIS collections. In each figure, the average number of bits per pointer is plotted as a function of q_0 and this is why the curves for "$\gamma - Code$", "*Golomb*" and "$\gamma - Golomb$" are horizontal lines.

A first observation is that the "$\gamma - code$" line is always above the "*Golomb*" line. This means that the γ code produces larger indexes than the Golomb code for the local Bernoulli model. These results confirm similar results presented in [11]. Another observation is that the "$\gamma - Golomb$" line is also always above the "*Golomb*" line. This means that if one of the two codes (γ or unary respectively) should be preferred for storing the values of q+1, then the unary code should be preferred. Indeed the unary code is preferred for the Golomb codes, as described in [11].

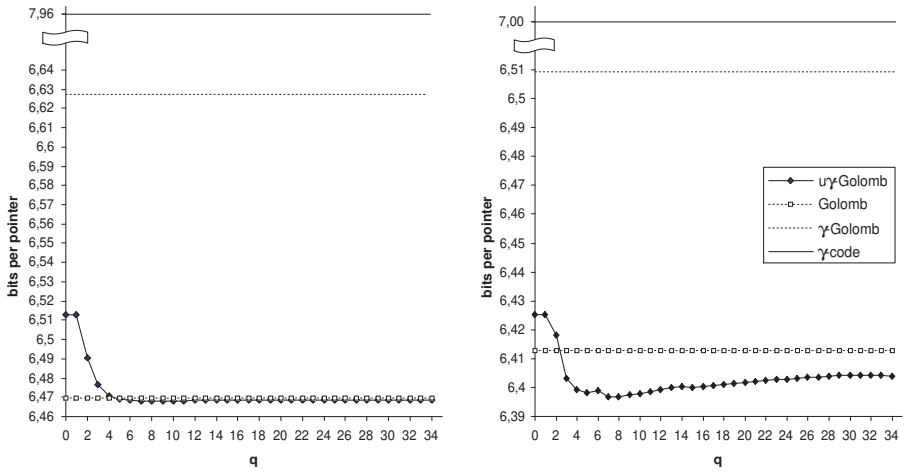

Fig. 1. The results for the LAT and the FBIS collections.

The results demonstrate that the compression achieved by "$u\gamma - Golomb$" is low for small values of q_0, it is maximized for values near 7 and 8, it keeps decreasing as q_0 increases beyond 9 and approaches asymptotically the compression of "*Golomb*". In the case of Figure 1 the difference between the "$u\gamma - Golomb$" and the "*Golomb*" curves is very small. The two curves overlap or remain very close to each other in the case of LAT and FBIS for values of $q_0 > 3$. This is probably because LAT and FBIS are uniform collections. Similar results have been obtained by using uniform synthetic collections generated by FINNEGAN [17]. In Figure 2 the average "$u\gamma - Golomb$" pointer utilizes nearly 0.1 fewer bits than the Golomb code pointer when $q_0 = 7$. This is a significant gain for our variation method. This is due to the fact that LATFBIS is a non-uniform collection, consisting of two different text database collections. Many words tend to appear often in one collection and not so often in the other. In this respect, we have relaxed the assumption for uniform distribution of the words in the documents. In fact, the more non-uniform a collection is, the more the two lower curves in Figure 2 tend to decline.

Having experimented with many other text database collections, uniform and non-uniform, we have obtained results similar to the ones presented. When $q_0 = 7$, $u\gamma - Golomb$ was found to achieve maximum compression and never perform worse than the plain Golomb code for the local Bernoulli model. In the case of non-uniform databases, the gain came without imposing any kind of additional overhead, including computational overhead. Namely, the plain Golomb and the proposed $u\gamma - Golomb$ with $q_0 = 7$ involve the same decompression CPU overhead.

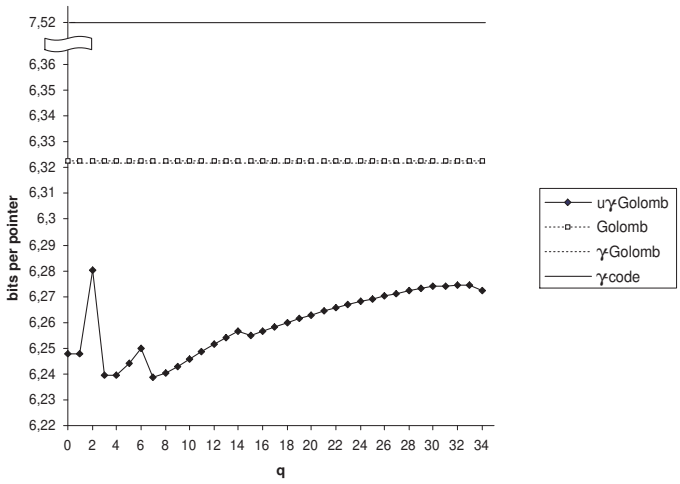

Fig. 2. The results for the LATFBIS collection.

5 Conclusion

In the present study we consider the issue of integer number encoding in the context of index compression for uniform and non-uniform document collections. The relevant research literature suggests that the Golomb code for the local Bernoulli model comprises the preferred choice for index compression.

We introduce $u\gamma - Golomb$, a new Golomb code variation, whereby a tunable threshold value q_0 (of the $q = \lfloor (x-1)/b \rfloor$ number used in the Golomb code for the local Bernoulli model) determines which d-gap values are (partially) encoded via the unary and which via the Elias γ-code scheme. Experimental runs of the proposed variation conducted against two TREC collections reveal Golomb code for the local Bernoulli model performance in the case of uniform collections and an increased compression efficiency for non-uniform collections. By construction, the $u\gamma - Golomb$ code does not introduce any additional CPU overhead during the encoding/decoding phase.

In the future stages of our research we intent to exploit the effect document reordering algorithms have to the performance of the proposed $u\gamma - Golomb$ code.

References

1. Blandford, D., Blelloch, G.: Index Compression through Document Reordering. Proceedings of the Data Compression Conference. (2002) 342–351
2. Bookstein, A., Klein, S.T., Raita, T.: Model based concordance compression. In Storer and Cohn. (1992) 82–91
3. Elias, P.: Universal codeword sets and representations of the integers. IEEE Transactions on Information Theory, Vol. IT-21. (1975) 194–203

4. Faloutsos, C.: Signature files: Design and performance comparison of some signature extraction methods. Proceedings ACM SIGMOD. (1985) 63–82
5. Faloutsos, C.: Signature Files. In: Frakes, W., Baeza-Yates, R. (eds): Information Retrieval: Data Structures and Algorithms. Prentice-Hall, Englewood Cliffs, NJ. (1992) 44-65
6. Fox, E., Harman, D., Baeza-Yates, R., Lee, W.: Inverted Files. In: Frakes, W., Baeza-Yates, R. (eds): Information Retrieval: Data Structures and Algorithms. Prentice-Hall, Englewood Cliffs, NJ, Chapter 3. (1992) 28–43
7. Gallager, R.G., van Voorhis, D.C.: Optimal source codes for geometrically distributed alphabets. IEEE Transactions on Information Theory, Vol. IT-21. (1975) 228–230
8. Golomb, S.W.: Run-length Encodings. IEEE Transactions on Information Theory, Vol. IT-21. (1966) 399–401
9. Huffman, D.A: A method for the construction of minimum redundancy codes. Procedures IRE, Vol.40(9). (1952) 1098–1101
10. Moffat, A., Stuiver, L.: Exploiting clustering in inverted file compression. In Storer and Cohn. (1996) 82–91
11. Moffat, A., Zobel, J.: Paremeterised Compression for Sparse Bitmaps. 15th Ann Int'l SIGIR, Denmark (1992) 274–285
12. Scholer, F., Williams, H.E., Yiannis, J., Zobel, J.: Compression of Inverted Indexes for Fast Query Evaluation. SIGIR, Finland (2002) 222–229
13. Schuegraf, E.J.: Compression of large inverted files with hyperbolic term distribution. Information Processing and Managemant Vol 12. (1976) 377–384
14. Teuhola, J.: A compression method for clustered bit-vectors. Information Processing Letters, Vol 7(2). (1978) 308–311
15. Williams, H. E., Zobel, J.: Compressing Integers for Fast File Access. The Computer Journal, Vol. 42. (1999) 193–201
16. Witten, I.H., Moffat A., Bell T.C.: Managing Gigabytes. Compressing and Indexing Documents and Images. Academic Press (1999)
17. Zobel, J., Moffat, A., Ramamohanarao K.: Inverted Files Versus Signature Files for Text Indexing. ACM Transactions on Database Systems, Vol. 23. (1999) 369–410